Journal of Computing and Information Technology - CIT 20, 2012, 4, 265–276
doi:10.2498/cit.1002017

CLOLINK: An Adapted Algorithm for Mining Closed Frequent Itemsets

Adebukola Onashoga

Department of Computer Science, Federal University of Agriculture, Abeokuta, Nigeria.

Mining of the complete set of frequent itemsets will lead to a huge number of itemsets. Fortunately, this problem can be reduced to the mining of closed frequent itemsets, which results in a much smaller number of itemsets. Methods for efficient mining of closed frequent itemsets have been studied extensively by many researchers using various strategies to prove their efficiencies such as Apriori-like methods, FP growth algorithms, Tree projection and so on. However, when mining databases, these methods still encounter some performance bottlenecks like processing time, storage space and so on. This paper integrates the advantages of the strategies of H-Mine, a memory efficient algorithm for mining frequent itemsets. The study proposes an algorithm named *CLOLINK*, which makes use of a compact data structure named *L_struct* that links the items in the database dynamically during the mining process. An extensive experimental evaluation of the approach on real databases shows a better performance over the previous methods in mining closed frequent itemsets.

Keywords: frequent pattern growth, closed frequent itemsets, data mining, mining methods and algorithm, CLOLINK

1. Introduction

Frequent pattern mining is the identification of frequent itemsets from large database. Given a transaction database, frequent patterns can be found in those transactions. For example, running a large shop requires maintaining terabytes of customer transactions. Frequent patterns in this situation, for example, could be items that are frequently bought together. The main goal of frequent itemset mining is to identify all frequent itemsets, that is, itemsets that have at least a specified minimum support; the percentage of transactions containing the itemset [14]. The rationale behind using support is that only itemsets with high frequency are of interest to users. According to [14], "the practical usefulness of the frequent itemset mining is limited by the significance of the discovered itemsets".

Definition 1: (Frequent itemset)

Let $I = \{x_1, \ldots, x_n\}$ be a set of items. An itemset X is a subset of items, i.e., $X \subseteq I$. For the sake of brevity, an itemset $X = \{x_1, x_2 \ldots, x_m\}$ is also denoted as $x_1, x_2, \ldots, x_m$. A transaction $T = (tid, X)$ is a 2-tuple, where tid is a transaction identifier (i.e. customer identifier) and X, an itemset. A transaction $T = (tid, X)$ is said to contain itemset Y if $Y \subseteq X$. A transaction database TDB is a set of transactions in TDB containing itemset X, the support of an itemset X is the number of times it occurs in a transaction, denoted as $\sup(X)$. Given a transaction database TDB and a support threshold, min_sup, an itemset X is a frequent pattern, or a pattern in short, if $\sup(X) \geq$ min_sup.

Efficient algorithms for mining frequent itemsets are crucial for mining association rules. An example of algorithm for mining frequent itemsets is Apriori [18] algorithm, which suffered from two drawbacks: (1) multiple scans of a dataset to compute the frequency of itemsets; (2) high number of generated association rules. The bottleneck of Apriori method rests on the candidate set generation and test. Alternatively, [6] proposed a pattern growth algorithm called FP-growth, this algorithm is reported to be an order of magnitude faster than the Apriori algorithm. FP-growth uses the Apriori property, but instead of generating candidate sets, it recursively mines patterns in the database, which is already represented in a tree structure named FP-Tree. [14] proposed a UMining algorithm for mining frequent itemsets, which is based on a levelwise approach. It was noted by [14] that

depth-first search approaches, such as MAFIA [15] and FP-growth [16], have several advantages over level-wise approaches. Their future research considers whether the pruning strategies proposed in the UMining algorithm can be incorporated into an approach such as FP-growth. As pointed out by [9], FP-Tree loses its compactness on sparse datasets.

In classical frequent pattern mining algorithms, huge number of frequent patterns are generated, whereas among them exist redundant information. **Closed frequent pattern mining**, however, solves this problem by returning a succinct result with redundancies being removed. The following scenario below gives a good understanding of closed frequent patterns.

Suppose the frequent patterns generated are: {*bread, butter: 10*}; {*sugar, butter: 10*}; {*bread, sugar: 10*}; {*bread, sugar , butter: 10*}. Closed frequent pattern mining will return one itemset only: {*bread, sugar, butter: 10*}. This itemset, however, represents the complete information about the frequency of its three sub-itemsets.

Definition 2: (Closed frequent itemset)

A frequent itemset X is said to be a closed itemset if there exists no X' such that X' is a proper superset of X and every transaction containing X also contains X'. This simply means that a closed itemset X is an itemset that is not included in another itemset having the same support.

In a mathematical form, given the functions:

$$f(T) = \{x \in I \mid \forall t \in T, x \in t\},$$

which returns all the itemsets included in the set of transactions T, and

$$g(I) = \{t \in T \mid \forall x \in I, x \in t\},$$

which returns the set of transactions supporting a given itemset I (its tid-list), the composite function $f \circ g$ is called Galois operator or closure operator.

Definition 3: An itemset I is said to be closed iff

$$c(I) = f(g(I)) = f \circ g(I) = I.$$

Proposition 1 [1]: Given an itemset X and an item x,

$$g(X) \subset g(x) \Rightarrow x \in c(X).$$

Proof: Since

$$g(X) \subset g(x) \Rightarrow g(X \cup x) = g(X).$$

Therefore, if $g(X \cup x) = g(X)$ then

$$\begin{aligned} f(g(X \cup x)) = f(g(X)) &\Rightarrow c(X \cup x) = c(x) \\ &\Rightarrow x \in c(x). \end{aligned}$$

Given an itemset X, by exploiting proposition 1, an item x can be determined whether it belongs to $c(X)$ or not. This proposition is used by most algorithms to calculate closures incrementally. Another important proposition, also by Claudio et al. (2004) is:

Proposition 2: Given two itemsets X and Y, if $X \subset Y$ and $\sup(X) = \sup(Y)$ i.e. $|g(X)| = |g(Y)|$, then $c(x) = c(Y)$.

Proof: If $X \subset Y$, then $g(Y) \subset g(X)$. Since $|g(Y)| = |g(X)|$ then

$$\begin{aligned} & g(Y) = g(X), \\ & g(X) = g(Y) \\ \Rightarrow & f(g(X)) = f(g(Y)) \\ \Rightarrow & c(X) = c(Y). \end{aligned}$$

Proposition 2 is to check for any duplicates.

According to [1], it is intuitive that the closure operator defines a set of equivalence classes over the lattice of frequent itemsets: two itemsets belong to the same equivalence class if they have the same closure, i.e. their support is the same and is given by the same set of transactions. Knowledge about closed frequent patterns is interesting and useful when efficient algorithms are used. The efficiency of any algorithm involves the processing time and the space utilized. Several researchers [9, 13] have also adapted existing algorithms in mining closed frequent, frequent sequential patterns in order to provide an efficient closed pattern mining algorithm.

In this paper, an adaptation of the H-Mine algorithm [5] is proposed for mining closed itemsets. H-Mine is a memory based pattern growth algorithm for mining frequent itemsets. It is tested to be more space and memory efficient than pattern growth methods like FP-growth [16] and TreeProjection, when running sparse and very large datasets [5]. This paper presents a pattern growth algorithm named CLOLINK ("CLO" from the closed itemsets being mined and "LINK" from the structure which makes

use of node-link to access items) for mining the complete sets of closed frequent itemsets, by using a data structure named L_struct (using the hashtable class in Visual basic.NET). CLOLINK is experimented on different datasets and then compared with CHARM, a tree projection algorithm. Experimental results revealed that CLOLINK is a scalable algorithm on any size of datasets by using a limited space in storing closed patterns. It also eliminates the bottleneck of the FP-growth and does not incorporate the features of candidate set generation.

The remaining part of this paper is organized as follows: Section 2 details the approaches used in mining closed frequent itemsets while Section 3 presents the steps in the proposed algorithm with an elaborate illustration using a sample database. Each of the steps described is followed by its comparison with H-Mine algorithm. Section 4 discusses the implementation and performance study of the algorithm. Section 5 concludes the paper and gives an extension of future work.

2. Review of Related Works

The strategies developed to speed up the process of mining closed frequent itemsets can be divided into two categories. The first is the candidate generation and test approach. Algorithms in this category include Apriori e.g. A-Close [2] which runs very slowly on long pattern data sets because of the huge number of candidate itemsets it has to generate and test. [7]. A-Close is the first algorithm for mining closed itemsets based on the Apriori heuristic, but looks for frequent closed itemsets and prunes the frequent itemsets that are not closed. The major cost of the A-Close is from two aspects: (1) it has to generate a lot of candidates and scan the transaction database again and again to count candidates; and (2) in the last scan to compute closures, there could be a large number of surviving frequent itemsets. For each transaction, the intersection with each surviving frequent itemsets is done. This makes the closure computation quite costly.

The second approach is Pattern growth which avoids the candidate generation and test by using the pattern growth algorithms e.g. Tree Projection, FP-growth (Frequent Pattern-growth) and so on. It also uses the Apriori property, but instead of generating candidate sets, it recursively mines patterns in the database counting the support for each pattern. Algorithms in this category make use of non-linear structures to store their databases which is more complicated when traversing [8], for example FP-Tree for FP-growth and lexicographical tree for CHARM.

The authors of FP-growth proposed CLOSET [4] for mining closed frequent patterns. This algorithm inherits from FP-growth, the compact FP-Tree data structure and the exploration technique based on recursive conditional projections of the FP-Tree. Frequent single items are detected after a first scan of the dataset, and with another scan, the pruned transactions are inserted in the FP-Tree stored in the main memory. Despite the efficiency of this FP-growth, if the database is huge, the FP-tree will be large and the space requirement for recursion is a challenge [3].

CHARM, a Tree Projection algorithm for finding closed frequent itemsets is proposed by [10]. CHARM performs a bottom-up depth first browsing of a prefix tree of frequent itemsets built incrementally. As soon as a frequent itemset is generated, its tid-list is compared with those of the other itemsets having the same parent. When two tid-lists are equal, or one includes the other, the associated nodes are merged since the itemsets surely belong to the same equivalence class. Itemset tid-lists are stored in each node by using the diff-set technique [11].

Mao in [12] finds out that CHARM scales much better than the FP-growth and Apriori based algorithms. So, in this context, CHARM is considered to be better. In an experimental evaluation by [7], CLOSET is unable to handle long biological datasets because of two reasons. First, the FP-Tree is unable to give good compression for long rows. Second, there are too many combinations when performing column enumerations.

3. Algorithm Design

This section presents the complete description of the proposed algorithm. It should be noted that steps 1 and 2 of this algorithm are the same as in the H-Mine algorithm. The major integration of H-Mine is in step 3 where the closed itemsets are mined.

3.1. Description of the Proposed Algorithm

Algorithm: Mining closed frequent patterns integrating the pattern growth method.

Input: Transaction database, TDB which consists of a set of items.

Output: Closed frequent itemsets.

Method: The algorithm can be divided into several steps as below:

Step 1 – Find frequent items

In this step, the transaction database is scanned once. During this scan, the count for each item is taken. In the meanwhile, their counts are now compared with the minimum support threshold to generate the frequent items. Those items that do not meet up with the minimum support are considered infrequent and hence, discarded.

For each itemset of a transaction, let $Freq(X)$ be the frequent-item projection of itemset X which includes only the frequent items found in each transaction. The order should be left as in the original database; the ordering is time efficient, since the objective of the study is to reduce the processing time. If multiple transactions share an identical frequent itemset, they can be merged into one with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are sorted according to an original fixed order.

To illustrate this step, see the example below:

Example 1: Using Table 1, let the first two columns be the transaction database, *TDB*, and setting the minimum support to 2. The frequent-item projection is shown in the third column.

Transaction ID	Items	$Freq(X)$
10	a, c, d, e, f	a, c, d, e, f
20	a, b, e	a, e
30	c, e, f	c, e, f
40	a, c, d, f	a, c, d, f
50	c, e, f	c, e, f

Table 1. The Transaction database (to be used as the running example).

By scanning the database once, the complete set of frequent items $(a : 3, c : 4, d : 2, e : 4, f : 4)$ are found and the output which forms the *F_List*. Note that item b has been pruned because its support is 1 and does not meet up with the $\min\_\sup = 2$.

Following the order of the *F_List*, the complete set of closed frequent patterns can be partitioned into 5 subsets as follows:

(1) those containing item a (2) those containing item c, but no item a (3) those containing item d, but no item a nor c (4) those containing item e, but not item a nor c nor d (5) those containing only item f.

Step 2 – Construct the *L*-struct for the frequent-item projection in Table 1

Definition 4: *L*-struct is a data structure defined below:

(1) It consists of a Lookup table, L (user-defined) such that every occurrence of a frequent item is stored in an entry with three fields: an *item-id*, the *support count* and a *node-link*. There is also a structure called *TransLink* which constitutes one for each of the frequent-item projections with two fields: an *item-id* and a *node-link*, and the other one identifying the first by its Trans-id, (denoted as T_j, where T represents each structure and j denotes each transaction) [see Figure 1].

(2) When the frequent-item projections are loaded into memory, those with the same first item (in the order of *F_List*) are linked together as a queue, and the entries in the Lookup table as the head of the queue.

The proposed algorithm uses the concept of data structure proposed in H-Mine, though there is a difference in the way it builds and adjusts links in other structure on identifying the closed itemsets.

Step 3 – Mining the *L*-struct to generate closed frequent patterns

Based on this definition, the remaining mining process can be performed on the $L-struct$ only without referencing any information in the original database. After that the sets of frequent patterns will be generated, each of these mined one by one to generate the closed frequent patterns as follows:

- After scanning the *TDB* once and having collected the frequent item projections of each

transaction, create a table-like structure labeled "$L - struct$" as described in Definition 4. Its support count registers the occurrence of each frequent item in the transaction database.

- Following the subsets of the frequent items, to mine the i-projected database (where i represents each frequent item), create a new structure that registers only the postfix items of each of the subsets, L_i-Lookup table. The support count of these items records the occurrence of the corresponding item with i in each frequent item projection. Then do the following:

  Let the sorted items in the L_i Lookup table be $[i \mid I]$, where I denotes each item in the postfix item-list. Call $insert\_struct([i \mid I])$, The function insert_struct($[i \mid I]$) is performed as follows:

  Check the occurrence of I in the L_i Lookup table, if I occurs with i in the T_j table, increment its count by 1. In this case, the set of locally frequent items i.e. the items appearing at least the defined min_sup times in the i-projected database is found. Then each I will now be combined with i to form iI_1, iI_2, $\ldots$ iI_n, where n is the number of frequent patterns generated from L_i. After getting each iI_k (where $k = 1, n$), build up links for L_i.

  Note: the i-projected database consists of all the frequent-item projections containing item i.

- Going by the definition of closed frequent patterns in Section 1, the set of frequent patterns obtained (including i itself) from the above algorithm are checked to generate closed frequent patterns. The procedure is:
  - First check which of these itemsets is closed, if there is one, then this is the output, while Lookup table is created for the remaining ones so far as they are frequent. This Lookup table i.e. L_{iIk} (where $k = 1, n$) will be drawn using the L_i Lookup table. The same process for finding frequent patterns is applied and then check for their corresponding closed itemsets. This continues until all possible combinations have been found.
  - After the closed frequent itemsets containing item i are found, the i-projected database i.e. i-queue is no longer needed in the remaining of mining. To traverse the i-queue once more, each frequent item projection in the queue is appended to the queue of the next item in the projection following i in the F_List.

The following pseudocode shows the procedure that handles this part:

```
Procedure CloMine(y, z, s)
// for generating closed frequent itemsets
y := the projected item being mined
z := concatenation of item being mined and its
     postfix
tid := transaction identifier
s := support
Add y to set of frequent itemset, F
For all frequent itemsets, F
  If s(F_i) = s(F_j) [where i = 1, n and j = 2, n]
    If tid(F_i) = tid(F_j)
      Add F_i and F_j to closed itemset
    Else
        For all itemset k ∈ F_i and k ∈ F_j
          If size(F_i) > size(F_j)
            Delete F_i from list
          EndIf
        EndFor
    EndIf
  Else
    Add F_i to closed itemset
  EndIf
EndFor
```

In comparison with H-Mine, CLOLINK checks on the closed itemsets on every build of the structure for the i-projected database. $L-struct$ does not store any other items aside the closed frequent itemsets generated. The $Construct()$ function checks and prunes out all duplicated itemsets. CLOLINK dynamically adjusts the links in the mining process.

3.2. Illustration

The general idea of CLOLINK is shown using the same transaction database in Table 1. Following the highlighted steps, the CLOLINK algorithm is illustrated using Table 1 in Example 1 (the same table used for illustration of algorithms in referenced articles). On getting the frequent-item projection, Figure 1 shows the $L - struct$.

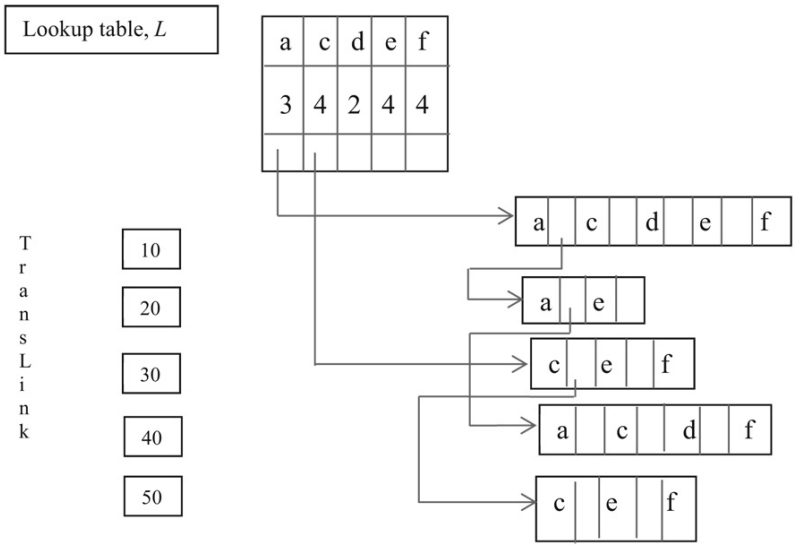

Figure 1. L − struct.

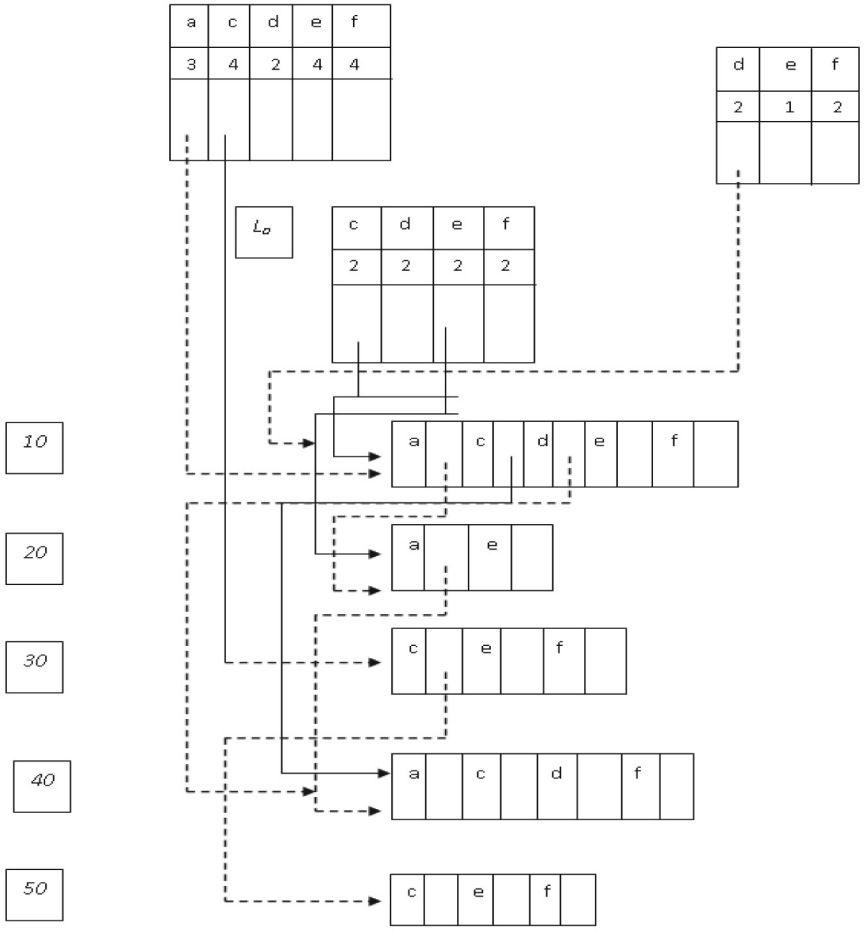

Figure 2. Lookup table, L_a.

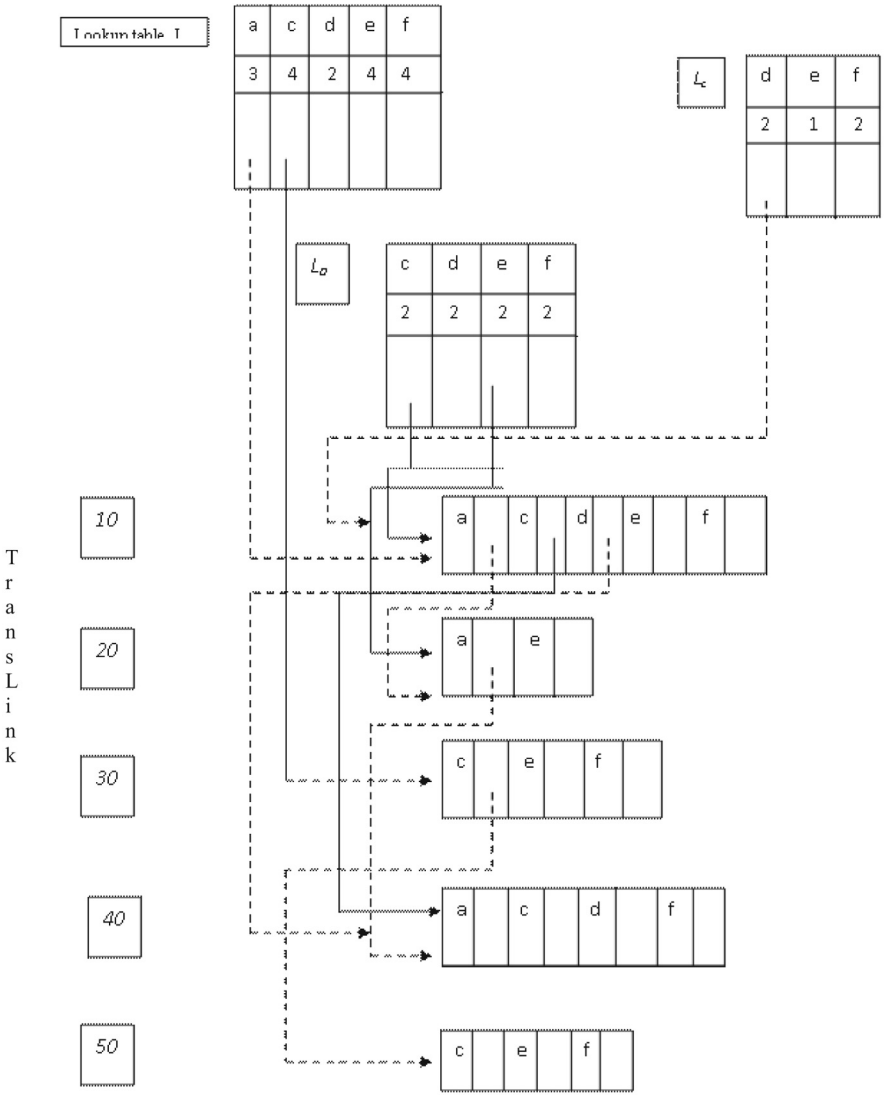

Figure 3. Adjusted node-links after mining the a-projected database and L_c Lookup table.

To mine the a-projected database, create a-Lookup table, L_a as shown in Figure 2. In L_a, every frequent item except for a itself, has an entry with the same three fields as L. The support count in L_a records the support of the corresponding item in the a-projected database. For example, item c appears twice with a in the a-projected database, thus the support count in entry c of L_a is 2.

By traversing the a-queue once, the set of locally frequent items, i.e., items appearing at least 2 times, in the a-projected database is found, which is $\{c : 2, d : 2, e : 2, f : 2\}$. This scan outputs frequent patterns $\{ac : 2, ad : 2, ae : 2, af : 2\}$. A link is now built for L_a table as shown in Figure 2. To check for closure itemsets, item a is contained in 3 transaction-ID i.e. $[10, 20, 40]$, whereas, ac, ad, af are not contained in same transactions as a, thus $\{a : 3\}$ is closed. It remains to check whether the frequent patterns are closed.

Similarly, the process continues for the ac-Lookup table. This output items d and f with same supports: 2 and are thus considered to be frequent, and then $\{e : 1\}$ which is infrequent. The frequent patterns $\{acd : 2, acf : 2\}$ are generated, which are now checked for closed patterns. Since acd is contained in the same transaction as cd, as and acd being a larger set, it is then said to be closed.

Likewise, the itemset acf is also closed in ac. This outputs only $\{acd : 2, acf : 2\}$ as closed itemsets. Each of these will now be checked to see if there exists a larger set by dynamically changing its link for acd and acf.

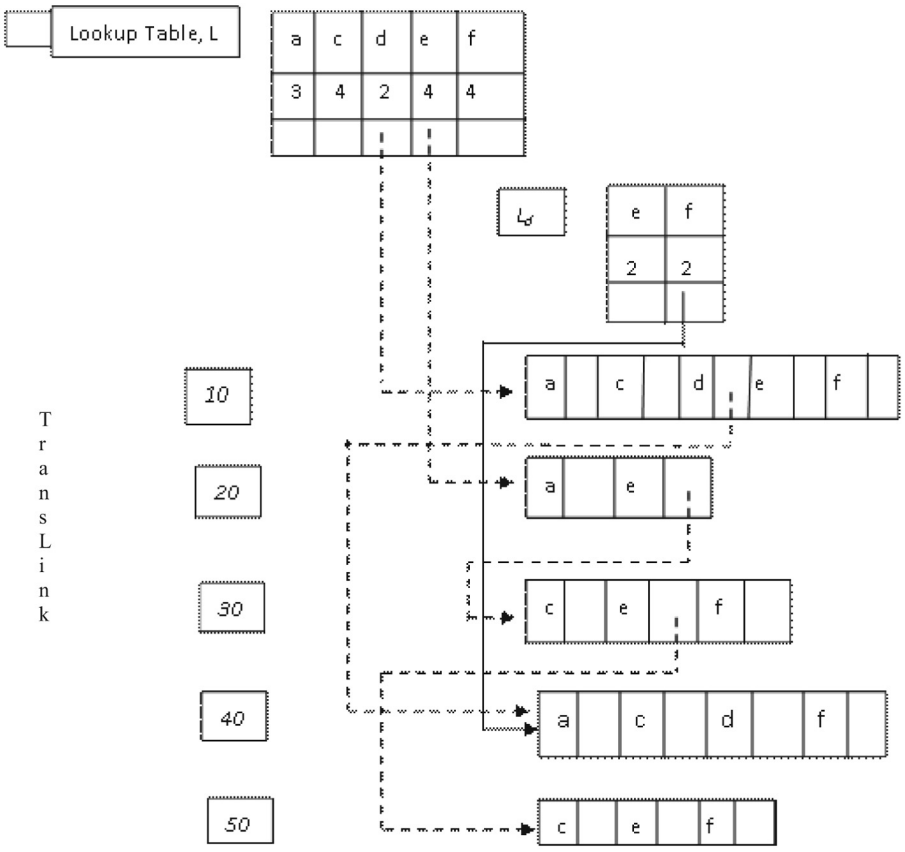

Figure 4. Adjusted node-links after mining the c-projected database and Lookup Table, L_d.

For acd-projected database, there are $\{e : 1, f : 2\}$, only f is frequent and thus $\{acdf : 2\}$ is output as frequent pattern. Since there is no larger set of it, it is considered to be closed.

For acd-projected database, d has already been considered and there is no link for item e, thus $acf : 2$ is output as closed itemset. The same process continues for ad-projected database using the L_a table, then the recursion backtracks to find patterns containing a and d, but no c, (item c has been considered in the mining of ac), this outputs $\{e : 1, f : 2\}$, only item f is frequent with ad. Thus, pattern $\{adf : 2\}$ is frequent. To check whether it is closed or not, it occurs in the same transaction as ad and there is no larger set of it, so it is considered to be closed.

For ae, from L_a table, only item f will be considered whose support is 1 and thus not frequent. Hence, itemset $\{ae : 2\}$ is closed (note: items c and d have been considered).

There will be no link for af in the L_a table, since all other items have been considered, thus considered closed.

The overall output from the a-projected database includes $\{a : 3, adf : 2, ae : 2, acdf : 2, acf : 2, af : 2\}$.

After the closed frequent patterns containing item a have been found, the a-projected database i.e. q-queue is no longer needed in the remaining of the mining.

The next step is to mine the c-projected database, i.e. find closed itemsets containing item c, but no a, using the same Lookup table, L, the lookup table for c-projected database (Figure 3) is drawn which includes only the postfix items of c, but no a.

To mine all frequent patterns containing item c but no a other subsets of frequent patters, there is a need to insert all the projections in the queue to the proper queues. By traversing the a-queue is appended to the queue of the next item from in the projection i.e. c as shown in Figure 3.

For the c-projected database, on creating the L_c table, the set of frequent items e, d and f is found. This outputs $ed : 2$, $ce : 3$, and $cf : 4$ (i.e. d appears twice with c, e thrice with c and

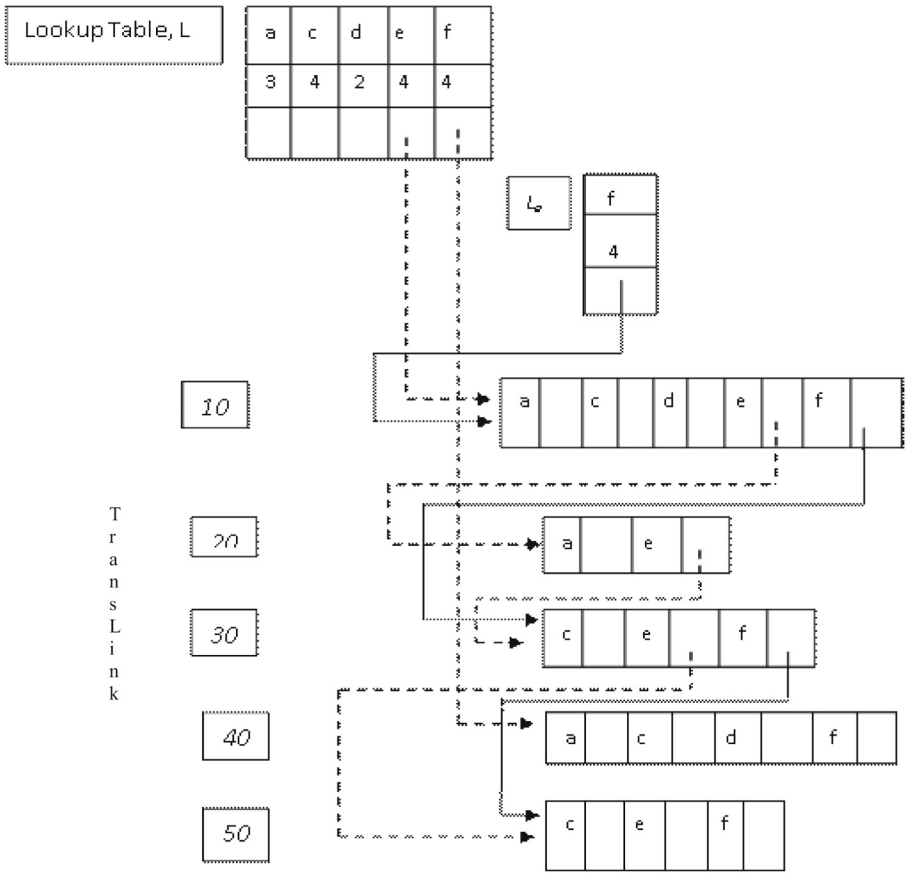

Figure 5. Adjusted node-links after mining the d-projected database and Lookup Table, L_e.

f four times with c) as frequent patterns, a link is now built for L_c table. To check for closed itemsets, item c is contained in 4 transactions i.e. Trans-Id: 10, 30, 40, 50 and only $\{cf : 4\}$ is contained in the same transactions unlike cd and ce. Thus $\{cf : 4\}$ is output as close itemset. To check any combination of cd and ce that can be closed, lookup tables for cd and ce are created i.e. L_{cd} and L_{ce} table.

For L_{cd}-projected database, using the L_c lookup table, items e and f are checked with cd with supports of 1 and 2 respectively. Since item f is the only one frequent with cd, thus $cdf : 2$ is frequent and it is contained in the same transaction as $cd : 2$.

For ce-projected database, only $f : 3$ is frequent and thus $\{cef : 3\}$ is frequent and thus $\{cef : 3\}$ is output as frequent pattern. Since every transaction containing ce also has f, there is a need to check whether there is any larger set than $cef : 3$ and it is discovered that there is no larger set of it. Thus, it is considered to be closed, i.e., $cef : 3$.

To check if there is any combination of cdf and cef that are closed, the link in the L_{cd} and L_{ce} will be changed dynamically. For the cdf - projected database, only item e is to be checked and it has been considered infrequent in cd, there is no link for it.

Conclusively, in the c-projected database the following patterns are derived as closed: $\{cdf : 2, cef : 3, cf : 4\}$.

To check the d-projected database, only items e and f will be considered without a and c. By traversing the d-queue once, the locally frequent item is $\{f : 2\}$ (Note $e : 1$ is not frequent and thus will not be considered). This scan outputs frequent pattern $(df : 2)$ and builds up link as shown in Figure 4. To check for the closed frequent pattern, item d is contained in the same transaction as df, but there is no larger set than $df : 2$, thus picked as a closed itemset.

There cannot be any combination of df that could be closed because on checking Figure 4 , there is no link, thus the mining completes here.

Therefore, the d-projected database has only df as a closed itemset.

As for the e-projected database, create the L_e table which contains only item f, but no a nor c nor d as shown in Figure 5. On checking the L_e table, the only locally frequent item is $f : 3$, the output $ef : 3$, as the frequent pattern. To check for the closed pattern, since item e occurs in 4 different transactions where ef does not occur, the closed pattern will be $e : 4$.

Frequent pattern ef is now checked to see if it has any combination that can be closed. On checking the L_e table, ef has no projected database, thus considered as closed itemset.

The closed itemset for the e-projected database includes $e : 4$ and $ef : 3$.

The remaining partition to be mined is that containing only item f, i.e. the f-projected database. On trying to create a lookup table for f, it is discovered that f does not have any postfix items, thus the mining completes on generating only $f : 4$ as a closed frequent itemset.

The last scan is to check for any duplication [1] in the following closed patterns generated: $\{a : 3\}$, $\{acdf : 2\}$, $\{af : 2\}$, $\{acf : 2\}$, $\{ae : 2\}$, $\{cf : 4\}$, $\{cdf : 3\}$, $\{cef : 3\}$, $\{df : 2\}$, $\{e : 4\}$, $\{ef : 3\}$, $\{f : 4\}$. On cross-checking, it is discovered that itemsets: $\{acdf : 2\}$, $\{adf : 2\}$, $\{acf : 2\}$, $\{cdf : 2\}$, $\{df : 2\}$ all occur in the same transactions where $\{acdf : 2\}$ is the largest set of all, thus picked as the closed frequent itemsets. Also $\{ef : 3\}$ occurs in the same transaction as $\{cef : 3\}$, thus $\{cef : 3\}$ is picked as closed itemset. Likewise, itemsets $\{cf : 4\}$ and $\{f : 4\}$ both occur in the same transaction, but $cf : 4$ being a larger set, is picked as a closed frequent itemset.

The overall output of the closed itemsets generated from the transaction database after removing duplication [1] is the set $\{a : 3\}$, $\{acdf : 2\}$, $\{ae : 2\}$, $\{cf : 4\}$, $\{cef : 3\}$, $\{e : 4\}$ which is the same closed pattern generated by A-Close, CLOSET and CHARM.

The efficiency of the algorithm, as highlighted below, comes from the comparison with other closed frequent pattern mining methods:

- First, the complete search can be done in one database scan by constructing the Lookup tables at all levels simultaneously and dynamically changing the links.
- Also, it can be seen that the transaction identifiers in the structure make it easy to see where each item occurs.
- Lastly, CLOLINK confines its search in a dedicated space. All information needed can be obtained from the structure, unlike with some other algorithms where you have to be checking through the database on every scan.

4. Implementation and Performance Study

All the tests were performed on a 733MHz Pentium III PC, WITH 128 MB RAM and 20 GB HD running Microsoft Windows XP. CLOLINK is written in Microsoft VISUAL Basic.NET.

In this paper, the runtime is the CPU time (starting from the construction of the data structure to the generation of patterns) and the support is the absolute occurrence frequency. The databases used are:

i. a telecommunication database from a PABX telecommunication switch of historical life of 7 months with size 11.8 MB. It is represented as $T60I35D660k$, where T represents the number of Callers' names (i.e. 60); I represents the numbers of areas called (i.e. 35); and D represents the number of records in the database (i.e. 660,000).

ii. a customer – transaction database from a supermarket in Nigeria of historical life of 4 months with size 3.1 MB. It is represented as T28I7D12K, where T, in this case, is the number of the customer-identifiers (i.e. 28), I is the number of items bought per transaction (i.e. 7) and D is the number of records in the database (i.e. 12,498).

Figure 6 shows the flexibility of the proposed algorithm. With the growth of pattern length the support threshold is decreasing. As the support threshold goes down, there is a change in the number of patterns generated. The difference between the numbers of patterns generated in situations of applying various support constraints is also depicted.

Figures 7 and 8 show the comparison of runtime with CHARM on $T60I35D660k$ and $T28I7D12K$

databases respectively. It can be seen that at high support threshold, CLOLINK is a bit faster than CHARM and at very low support threshold the CLOLINK has difference from CHARM and thus performs better.

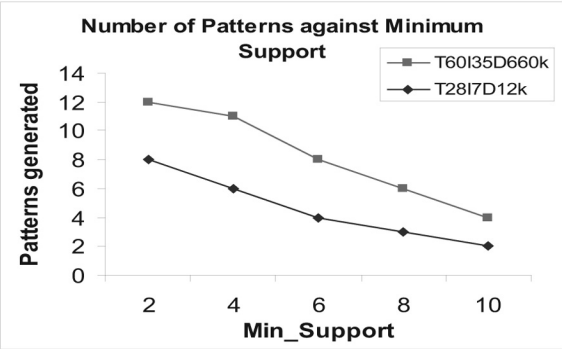

Figure 6. The Minimum support with the number of closed patterns generated.

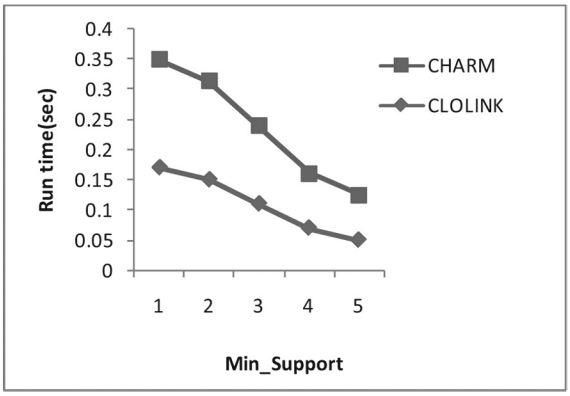

Figure 7. The Minimum support with the runtime (in seconds) on $T60I35D660k$.

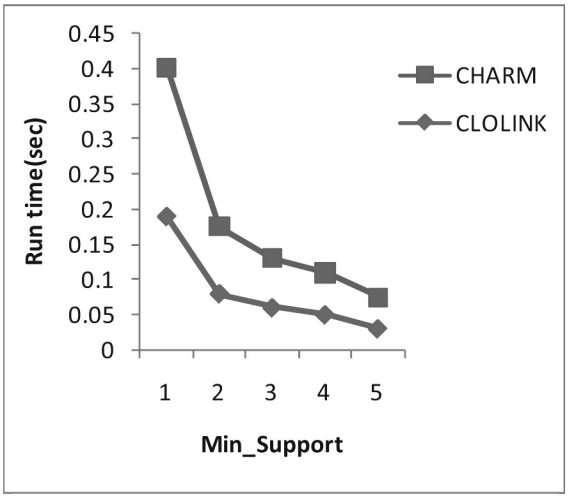

Figure 8. The Minimum support with the runtime in seconds on $T28I7D12K$.

In order to calculate the percentage difference between the two algorithms, the following equation is used on both $T60I35D660K$ and $T28I7D12K$.

$$\sum \left(\frac{\left(\frac{Runtime(CLOLINK) - Runtime(CHARM)}{Runtime(CHARM)} \right)}{N} \right) \cdot 100$$

Based on the above equation, it is discovered that on $T60I35D660k$, CLOLINK is by 17.12% better than CHARM. Also, on $T28I7D12K$, CLOLINK is better by 17.92%. Conclusively, CLOLINK performs better than CHARM at an average of 18%.

5. Conclusion

This paper adapted the ideas of the H-Mine algorithm to find closed frequent itemsets because of its efficiency. This adaptation allows a presentation of a data structure called *L_struct* and a memory efficient algorithm called CLOLINK for discovering closed frequent itemsets. The algorithm needs to scan the database only once. In the experimental comparison of the performance of CLOLINK against CHARM on $T60I35D660k$ and $T28I7D12K$, the result shows that the proposed algorithm outperforms CHARM on this database for the given ranges of support levels. A major distinction of CLOLINK from the previously proposed methods for mining closed frequent itemsets is that CLOLINK readjusts its link in mining different projected databases. It integrates the advantages of the previously proposed effective strategies to achieve a higher performance. Furthermore, it is shown that the interestingness of the closed frequent itemsets can be shown by pushing various support constraints.

6. Future Work

Real-time mining

Real-time databases are concerned with the storage and collection of data under the time collection constraints and time-constrained transactions. Integrating closed frequent itemsets mining into a real time database system is an active research area where hidden and on-line knowledge could be discovered on a real-time basis.

References

[1] C. LUCCHESE, S. ORLANDO AND R. PEREGO, Mining frequent closed itemset without Duplicates generation,*Proc. of the 2004 ACM-SIGMOD Int. Conf. on Management of Data* (2004).

[2] N. PASQUIER, Y. BASTIDE, R. TAOUIL, AND L. LAKHAL, Discovering frequent closed itemsets for association rules, *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, (1999).

[3] J. PEI, J. HAN, B. MORTAZAVI-ASL, H. PINTO, Q. CHEN, U. DAYAL, AND M. C. HSU, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, *Proc. of the 2001 Int. Conf. on Data Engineering (ICDE'01)*, (2001).

[4] J. PEI, J. HAN, AND R. MAO, CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, *Proc. of the 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)*, (2000).

[5] J. PEI, J. HAN, H. LU, S. NISHIO, S. TANG, D. YANG, H-Mine: Hyper Structure Mining of Frequent Patterns in Large Databases, *ICDM 2001 Proceedings of the 2001 IEEE International Conference on Data Mining*, (2001).

[6] J. HAN, J. PEI, AND Y. YIN, Mining Frequent Patterns without Candidate Generation, *Proc. of the 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00)*, (2000).

[7] P. G. RAJ AND G. S. YUDHO, ITL-MINE: Mining Frequent Itemsets More Efficiently, *Proc. of the 2002 Int. Conf. on Knowledge Discovery and Data Mining (KDD'00)*, pp. 35–49, (2000).

[8] L. SEYMOND, *Schaum's Outline Series: Theory and Problems of Data Structures*, McGraw-Hill Inc., Singapore, 1986.

[9] M. BEN HADJ HAMIDA AND Y. SLIMANI, A Patricia-Tree Approach for Frequent Closed Itemsets, *World Academy of Science, Engineering and Technology*, 2005.

[10] M. J. ZAKI AND C. HSIAO, CHARM: An efficient algorithm for closed itemset mining, *In SDM'02*, April, (2002).

[11] M. J. ZAKI AND K. GOUDA, Fast vertical mining using diffsets, *Technical Report 01-1*, RPI, 2001.

[12] R. MAO, Adaptive-FP: An efficient and effective methods for Multi-level Multi-dimensional frequent pattern mining, *M.Sc. Computer Science thesis*, Simon Fraser University, (2002).

[13] U. NIRANJAN, R. B. V. SUBRAMANYAM, V. KHANAA, An Efficient System Based on Closed Sequential Patterns for Web Recommendation System, *International Journal of Computer Science Issues*, Vol. 7, Issue 3, No. 4, May 2010.

[14] H. YAO, H. J. HAMILTON, Mining itemset utilities from transaction databases, *Data & Knowledge Engineering (Elsevier)*, Vol. 59, pp. 603–626, (2006).

[15] D. BURDICK, M. CALIMLIM, J. FLANNICK, J. GEHRKE, T. YIU, MAFIA: a maximal frequent itemset algorithm, *IEEE Transactions on Knowledge and Data Engineering*, 17(11), pp. 1490–1504, (2005).

[16] J. HAN, J. PEI, Y. YIN, R. MAO, Mining frequent patterns without candidate generation: a frequent-pattern tree approach, *Data Mining and Knowledge Discovery*, 8(1), pp. 53–87, (2004).

[17] F. PAN, G. CONG, A. K. H. J. YANG M. J. ZAKI, CARPENTER: Finding Closed Patterns in Long Biological Datasets, *SIGKDD '03*, August 2427, Washington, DC, USA, (2003).

[18] R. AGRAWAL AND R. SRIKANT, Fast algorithms for mining association rules, *Proc. of the 1994. Int. Conf. on Very Large Data Bases (VLDB '94)*, pp. 487–499, (1994).

Received: November, 2011
Revised: December, 2012
Accepted: December, 2012

Contact address:
Adebukola Onashoga, Ph.D.
Department of Computer Science
University of Agriculture
P.M.B 2240
Abeokuta
Ogun State
Nigeria
e-mail: onashogasa@gmail.com

ADEBUKOLA ONASHOGA is a lecturer at the Department of Computer Science, Federal University of Agriculture, Abeokuta, Nigeria. She has a Ph.D. degree in computer science, with a focus on computer security and data mining. She has published in both international and local journals. She has attended and presented papers at reputable international conferences. Her current research interests include computer security, data mining and pervasive computing.